Z를 이용한 컴포넌트 명세 전략 및 방법

장종표, 김병기
전남대학교 전산학과
e-mail:{jpjang, bgkim}@chonnam.chonnam.ac.kr

Component Specification Method using Z

Jong-Pyo Jang, Byung-Ki Kim
Dept of Computer Science, Chonnam National University

요약

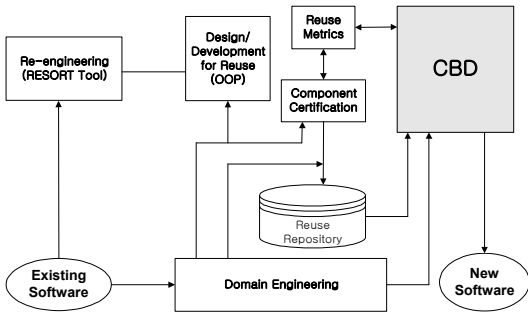
CBSE를 이루는 관련 기술들 중 하나인 소프트웨어 아키텍처는 시스템의 구조적 기술로서 시스템을 구성하는 컴포넌트와 그 컴포넌트들 사이의 상호작용을 기술한 것이다. 소프트웨어 아키텍처는 CBSE에서의 컴포넌트 사이의 조합에 대한 기술 및 방법론을 제공함을써, CBSE의 핵심 기술 중 하나로 자리잡고 있다.

본 논문에서는 컴포넌트 명세를 위해서 Formal methods와 ADL이 가지고 있는 장단점을 상호보완하하기 위하여 Z언어를 이용하여 컴포넌트, 커넥터, 전체 시스템 구성을 기술하는 3가지 전략과 이 전략을 기반으로 컴포넌트를 명세하는 방법을 제안한다.

1. 서론

소프트웨어 생산성이 사용자들의 서비스에 대한 요구를 만족시키지 못했고, 소프트웨어 품질이 향상되지 않았으며, 유지보수가 어려운 문제점을 가지고 있다. 이러한 위기를 해결하기 위한 한가지 대응책으로 최근 컴포넌트기반 소프트웨어 공학(Component-Based Software Engineering, CBSE) 혹은 컴포넌트웨어(Componentware)가 등장하였다.

[그림1]은 CBSE와 관련된 많은 기술들로 구성된 환경 모델이다[1].



[그림1] 관련 기술을 통합하는 CBD 환경[1]

CBSE를 이루는 관련 기술들 중 하나인 소프트웨어 아키텍처는 시스템의 구조적 기술로서 시스템을 구성하는 컴포넌트와 그 컴포넌트들 사이의 상호작용을 기술한 것이다. 소프트웨어 아키텍처는 CBSE에서의 컴포넌트 사이의 조합에 대한 기술 및 방법론을 제공함을써, CBSE의 핵심 기술 중 하나로 자리잡고 있다.[2]

본 논문에서는 컴포넌트 명세를 위한 3가지 전략과 이 전략을 기반으로 컴포넌트를 명세하는 방법을 제안한다.

본 논문의 구성은 다음과 같다. 제2장에서는 관련 연구로 각 ADL(Architecture Description Languag)과 ADL에서 공통적으로 기술하고 있는 부분을 정리하고, 아키텍처 명세에 정형방법이 필요한 이유와 정형 명세 언어인 Z에 대하여 살펴본다. 제3장에서는 컴포넌트 명세 언어와 정형 명세 언어가 서로 보완하는 특성을 이용한 컴포넌트 명세전략과 명세 방법에 대하여 제안한다. 제4장에서는 본 논문에서 제안한 컴포넌트 명세 전략 및 컴포넌트 명세 방법에 대한 결론 및 향후 연구방향에 대하여 제시한다.

2. 관련연구

2.1 ADLs(Architecture Description Languages)

2.1.1 Larch[3]

Larch는 대수 명세의 형태를 지닌 컴포넌트 명세 언어이다. Larch는 컴포넌트를 기술하기 위해서 LIL(Larch Interface Language)와 LSL(Larch Shared Language)를 동시에 지원하여, 기존의 프로그래밍 언어에 익숙한 개발자로 하여금 정형 명세 언어인 Larch에서 지원하는 여러 가지 검증 기법을 사용할 수 있도록 하고 있다. 개발자는 LIL을 통해 기존 프로그래밍 언어와 유사한 형태의 컴포넌트 인터페이스 명세를 하고, LIL에 사용된 명세 요소들에 대해 LSL을 기술하여 정형 명세를 완성하게 된다. Larch/Java는 Java 인터페이스를 기술할 수 있는 LIL을 지원하는 Larch의 한 형태로 기본적으로 Java 구현을 가정하고 있다.

2.1.2 RESOLVE[4]

RESOLVE는 다양한 구현 환경을 지원하기 위해 인터페이스언어와 구현 언어를 분리한 다중 차원 형태의 명세를 지원하고 있다. 따라서 하나의 인터페이스에 다양한 구현이 가능하여 유연한 개발을 지원한다. 또한 RESOLVE는 명세에서 기존의 다양한 구현 언어로의 변환을 유도하는 기법을 제시하고 있다. 예를 들어 RESOLVE/C++은 객체지향 프로그래밍 언어인 C++로 명세를 변환할 수 있는 기법 중 하나이다.

2.1.3 Darwin[5]

Darwin은 분산 소프트웨어의 아키텍처를 명세하는 것을 지원하기 위해 정의된 선언적 결합 언어이다. 특히, 제공되는 서비스와 요구되는 서비스들의 집합으로 이루어진 인터페이스에 의해 컴포넌트 타입을 기술한다. 또한 정적인 구조뿐만 아니라, 실행중에 변경되는 동적인 구조에 대한 명세를 지원한다. 그리고, Π-calculus를 기반으로 하여 컴포넌트간의 상호작용과 결합을 모델링한다. 그러나, 상호작용 패턴들은 그것을 제공하는 컴포넌트에 독립적으로 기술되지 못하는 스타일에 약점이 있다.

2.1.4 Unicon[6]

Unicon은 블랙박스 구현에 기초한 실행가능한 형상들을 구성하기 위한 도구들을 제공한다. 컴포넌트와 커넥터의 인터페이스가 이미 정의되어 있는 타입으로 정의된다. Darwin이 커넥터들의 비대칭적인 제공/요구되는 선언들을 통해 상호작용을 제공하는데 반해, Unicon은 명확하고 대칭적, 비대칭적인 커넥터들을 지원한다.

2.1.5 Rapide[7]

Rapide는 모델링 평가와 부분적으로 배열된 이벤트 집합들로서의 상호작용에 기초한 아키텍처 기술언어이다. 구현을 통하여 컴포넌트들의 행위적인 의미론을 정의하는 Darwin이나 Unicon과는 다르게, 관찰되거나 시작되는 통신 이벤트들의 집합에 대한 컴포넌트 타입들을 정의한다. 새로운 상호작용 타입들의 수용에 절절하지 못하고, 대칭적인 상호작용 패턴들을 지원하지 않는다.

2.1.6 Aesop[8]

Aesop는 구조적인 설계 환경들의 집합이다. 타입들의 객체지향 프레임웍을 통해 아키텍처 기술을 위한 용어를 제공한다. 컴포넌트, 커넥터, 포트, 역할, 형상과 바인딩으로 이루어진 기본적인 아키텍처 타입들의 서브타입들을 기술하고 있는 반면, 아키텍처 형상은 객체 인스턴스들의 상호연결된 집합으로 표현된다.

2.1.7 ACME[9]

ACME는 몇몇의 아키텍처에 관한 연구 그룹들의 공동노력으로 개발되었다. 기본적인 아키텍처 요소로는 컴포넌트, 커넥터, 시스템, 포트, 역할, 표현, 표현지도이다. ACME는 다른 언어들의 이점과 결합할 수 있는 툴들을 통해 공통적인 골격을 제공하는 것에 중점을 두었다.

2.1.8 SADL[10]

아키텍처 정제에 대한 정형적 기반을 제공하여 소프트웨어 아키텍처의 계층을 표현하기 위한 언어이다. 시스템의 행위보다 구조적 특성에 초점을 둔 언어로 시스템의 행위에 초점을 둔 기존의 아키텍처 기술언어와 상호 보완적으로 사용될 수 있다. ω-논리를 정형기반으로 하고 있고, 아키텍처를 기술하는 도구와 구체화의 정확성에 대한 분석의 도구를 지원한다.

2.1.9 WRIGHT[11]

WRIGHT는 구조적 환경과 구조적 스타일 모두의 기술을 위한 실제적인 정형 기초를 제공한다. 상호작용 패턴, CSP계열의 표기법을 사용하여 컴포넌트들의 추상적인 행위를 기술하는 능력, 시스템 인스턴스를 통한 술부를 이용한 스타일의 특성화, 그리고 구조명세의 일관성과 완벽성을 결정하기 위한 정적인 검사 집합으로 명확하고 독립적인 커넥터 타입을 사용한다.

2.2 ADL의 공통적인 기술[2]

2.2.1 컴포넌트의 기술

컴포넌트 기술 부분에는 시스템에 사용될 각각의 컴포넌트에 대한 행위가 기술된다. 컴포넌트 기술의 경우 전체적인 컴포넌트의 모든 행위가 아닌, 다른 컴포넌트와의 통신을 위한 부분적인 행위들이 기술되어 왔다. 컴포넌트의 기술은 또한 다음과 같이 두 가지 부분으로 구분할 수 있다.

첫째는 컴포넌트의 포트에 관한 부분이다. 포트는 컴포넌트가 실제 다른 컴포넌트와의 통신을 위한 부분으로 모든 이벤트들이 이 포트를 통하여 발생하고 받아들여지게 된다.

둘째, 이러한 포트들 사이의 관계에 대한 기술로써, 컴포넌트의 내부 행위 중에서도 포트사이에 일어나는 상관관계에 대한 기술이다. 이러한 기술을 통하여 실제 전체 컴포넌트를 분석시 시스템의 행위를 보다 정확하게 파악할 수 있다.

2.2.2 커넥터 기술

커넥터는 기존의 객체 지향 모델링에서 관계로써 간단하게 나타내어진 클래스간의 관계가 컴포넌트 사이의 통신 규약의 형태로 기술된 부분이다. 이 부분이 실제 아키텍처에서 특징적으로 이루어지고 있는 부분으로써 단순히 컴포넌트가 연관되어 있음을 표현하는 것 이외에 실제 두 컴포넌트의 상호작용의 행위를 명세한다.

이러한 통신 규약을 표현하기 위한 연구는 기존의 규약 검증을 위한 연구들에서 이루어진 정형 명세 기법들이 도입되었다. 예를 들어, CSP, POSET, π-calculus등이 이에 해당한다. 이러한 정형 명세 언어를 기반으로 함으로써, 아키텍처에 대한 많은 검증 및 분석을 할 수 있게 되었다.

2.2.3 전체 시스템 구성 기술

시스템을 이루는 각각의 컴포넌트와 커넥터들이 기술되면 이러한 요소들을 인스턴스화시키고 연결하여 실제 시스템을 구성하는 부분이 필요하다. 이러한 기술이 전체 시스템 구성 기술이다. 이러한 시스템 구성에 대한 기술로써, 각 구성요소에 대한 인스턴스화 선언과 커넥터와 컴포넌트 사이의 연결에 대한 기술이 이루어지게 된다. 이러한 과정을 통하여, 타입의 형태로 정의된 각각의 컴포넌트들과 커넥터 기술들의 재사용이 가능해 진다.

2.3 아키텍처 명세에 정형방법의 필요성[11]

정형방법은 강력한 분석력, 추상화, 그리고 구현의 세부사항으로부터의 독립성을 제공한다. 그러나, 커넥터와 스타일같은 기본적인 구조적 구성요소와 추상화를 제공하는데 약하다. 반면에, ADL은 복잡한 실세계 시스템을 구축하는 문제에 적용할 수 있는 컴포넌트와 커넥터를 위한 표기법을 정의함으로써 구조화 시키는데 매우 적합하다. 그러나, 현재 ADL은 정형방법이 가지고 있는 장점에 약하다. 즉, ADL은 분석적인 힘이 약하다. 그래서, ADL과 정형방법의 균형적인 접근 방법이 필요하다.

2.4 정형 명세 언어[12]

정형적 소프트웨어 명세서란 어휘, 구문, 의미 등이 정형적으로 정의된 언어로써 표현된 명세서이다. 정형 명세 언어는 자연어에 기초할 수 없기 때문에 정형적 의미 정의가 필요한 것이고, 수학에 기초를 둔다. 이런한 정형 명세를 사용함으로써 얻을 수 있는 잠재적인 기술적 잇점들은 다음과 같다.

·정형적 명세서의 개발은 소프트웨어 요구 사항과 소프트웨어 설계에 대한 고찰과 이해를 제공한다.
·정형화된 소프트웨어 명세서를 수학적인 개체로 표현하고, 수학적 방법론을 사용하여 분석될 수 있다.
·정형적 명세서는 자동적으로 처리될 수 있다.
·정형적 명세서는 구성 요소의 검사자가 적당한 검사를 할 수 있도록 도와주는 안내자 역할을 할 수도 있다.

[표1] 정형 명세 언어

접근방법	순차적 시스템	병렬 시스템
대 수 적 접근방법	Larch (1985) OBJ (1985)	Lotos (1987)
모델기반 접근방법	Z (1992) VDM (1980)	CSP (1985) Petri Nets (1981)

2.4.1 대수적 명세서

큰 시스템들은 대개 개별적으로 개발되는 부 시스템들로 구성되어 있고, 부 시스템은 또 다른 부 시스템을 사용한다. 따라서 명세서 프로세스의 본질적인 부분은 부 시스템의 인터페이스를 정의하는 것이다. 일단 인터페이스가 동의되고 정의되면, 부 시스템들은 개별적으로 개발될 수 있다. 대수적 접근은 특히 부 시스템의 인터페이스를 정의하는데 적합하다.

< SPECIFICATION NAME > (Generic Parameter)

sort <name>
imports <LIST OF SPECIFICATION NAMES>

informal description of the sort and its operations

Operation signatures setting out the names and the parameters to the operations defined over the sort

Axioms defining the operations over the sort

[그림2] 대수적 명세서의 형식

2.4.2 모델 기반 명세서

모델 기반 명세서는 시스템 명세서를 시스템 상태 모델로써 표현하는 정형적 명세서에의 접근법이다. 이 상태 모델은 집합이나 함수와 같이 잘 알려진 수학적 개체를 사용하여 구성된다. 시스템 모델의 상태에 어떻게 영향을 미치는가를 정의함으로써 명세서가 된다.

2.5 Z[13]

Z는 표준 집합론에 기반하는 정형표기이고, 시스템의 특성을 표현하기 위해서 양식화된 수학적인 표기를 사용한다. Z 명세는 스키마를 사용하여 구조화된다. 스키마는 묘사의 정형적인 부분을 포함할 뿐 아니라, 비정형적인 표현으로부터 정형 묘사를 분리하는 것을 도와준다. Z는 3가지 표기법을 가지고 있는 상호교환언어이다. 하나는 특별한 도식적인 기호들을 사용하고, 다른 2개는 전적으로 문자에 기반한다.

2.6 Object-Z[14]

Object-Z는 객체지향을 적용하기 위해서 정형명세언어 Z를 확장한 것이다. 이런 확장의 주요한 이유는 강화된 구조를 통해서 거대한 명세의 명확성을 향상시키는데 있다. Z 명세는 어떤 연산스키마가 하나의 특별한 상태스키마에 영향을 주는지를 추론하는 것은 모든 연산스키마의 서명들을 살펴보는 것을 요구한다. 커다란 명세에 있어서 이것은 실행 불가능하다. Object-Z는 하나의 상태스키마를 참조하는 개별적인 연산들을 제한함으로써 Z가 가지고 있는 극복하였다.

3. Z를 이용한 컴포넌트의 정형 명세

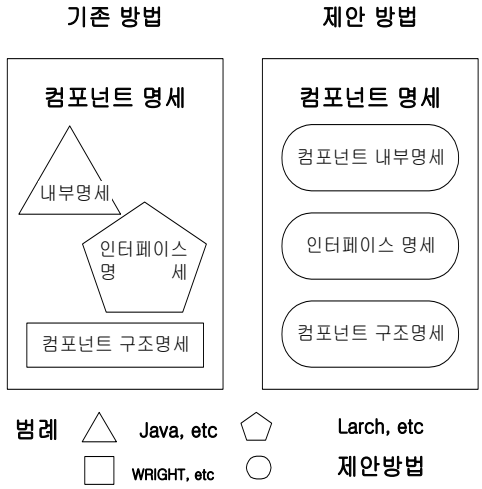
3.1 컴포넌트 명세를 위한 전략

[전략1] 정형 언어를 사용하여 컴포넌트를 명세한다.

소프트웨어 구조에 대한 접근방법인 형식 방법(Formal methods)과 구조 명세 언어 (Architecture Description Language, ADL)는 보충적인 장점과 단점을 가지고 있다[3]. Formal methods는 강력한 분석력, 추상화, 그리고 구현의 자세한 사항으로부터의 독립성을 제공하지만, 커넥터와 스타일과 같은 기본적인 구성요소들을 제공하지는 못한다. 반면에, ADL들은 컴포넌트와 커넥터에 대한 표기법을 정의함으로써 구조와 매우 잘 조화를 이룬다. 그러나, 정확히 형식화된 측면을 만드는 부분이 부족하다. 그래서, 두가지 접근 방법을 결합하는 것이 최선책이다. ADL로 아키텍처의 구조를 인식하고 Formal methods로 시스템의 특성들에 대하여 기술한다.

[전략2] 커넥터를 기본으로 하는 ADL을 확장하여 컴포넌트 내부의 계산과정을 표현한다.

대부분의 ADL에서 지원하고 있는 컴포넌트 기술에 관한 부분들은 아키텍처 명세에 나타나고 있는 컴포넌트와의 접합부인 커넥터를 기본으로하여 해당 아키텍처에서 사용하게 될 컴포넌트의 커넥터의 정의를 도와주는 제약을 기술하는데 치중되어 있다. 즉 컴포넌트 내부의 계산과정이나 커넥터를 포함한 컴포넌트 자체의 모델링이 미약한 상황이다. 그래서, 컴포넌트 내부의 계산과정이나 접합부를 포함한 컴포넌트 명세를 하고자 한다.



[그림3] 기존방법과 제안방법의 비교

[전략3] 컴포넌트의 종류별로 컴포넌트를 명세한다.

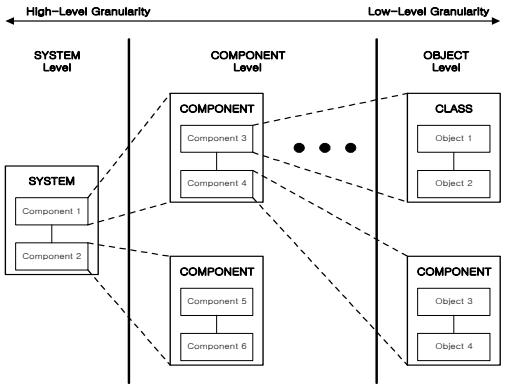
컴포넌트에 대한 정의는 다양하다. Rational사의 Philippe Krutchen은 “컴포넌트는 잘 정의된 아키텍처 상에서 어떠한 기능을 수행하는 시스템의 독립적이면서 대치 가능한 부분이다”로 정의하고 있다[4]. 이러한 컴포넌트는 구성요소의 세분정도

(granularity)에 따라서 다음과 같은 종류로 분류할 수 있다.

[표2] 컴포넌트 타입

종 류	정 의
System Type	서브시스템(컴포넌트 타입)의 집합으로 이루어진 컴포넌트
Component Type	여러 컴포넌트들의 집합으로 이루어진 컴포넌트
Object Type	최소단위인 object들과 그들 사이의 관계집합으로 이루어진 컴포넌트

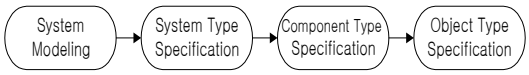
[그림4][표2]는 컴포넌트의 종류를 보여준다. 시스템 타입(System Type)에서는 전체 시스템 구성에 대한 기술을 중점적으로 표현하고, 컴포넌트 타입(Component Type)에서는 커넥터 중심으로 컴포넌트를 기술하고, 객체 타입(Object Type)에서는 컴포넌트 내부의 기술을 중점적으로 기술한다.



[그림4] 컴포넌트 세분정도에 따른 타입

3.2 Z를 이용한 컴포넌트의 정형명세

[전략3]에서 제시한 바와 같이 컴포넌트 구성요소의 세분정도(granularity)에 따라서 분류된 컴포넌트 타입에 의해서 높은 추상화로부터 세부적인 사항이 포함된 낮은 추상화로 명세한다. [그림5]는 각 단계의 명세를 도식화한 것이다.



[그림5] 컴포넌트 명세 단계

단계1에서는 시스템 모델로부터 시스템 타입 명세를 하는 단계로 전체 시스템에 대한 구성에 대하여 기술한다. 단계2에서는 시스템 타입 명세로부터 컴포넌트 타입 명세를 하는 단계로 커넥터를 중심으로 컴포넌트를 기술한다. 단계3에서는 컴포넌트 타입 명세로부터 객체 타입 명세를 하는 단계로 컴포넌트 내부를 중점적으로 기술한다.

[전략1]과[전략2]에 의해서 Z와 Object-Z를 이용한 컴포넌트는 다음과 같이 정의한다.

Component_name
Component Type
Component Definition Schema
Connector Definition Schema
Structure Definition Schema

[그림6] 컴포넌트 아키텍처

[그림6]은 컴포넌트 아키텍처를 나타낸다. 컴포넌트 아키텍처는 컴포넌트 타입, 컴포넌트 정의 스키마, 커넥처 정의 스키마, 구조 정의 스키마로 구성되며, 컴포넌트 타입은 시스템 타입, 컴포넌트 타입, 객체 타입중 하나이다.

Component Definition Schema
type definition
constant definition
state schema
initial state schema
operation schema

[그림7] Component Definition Schema

[그림7]은 컴포넌트 정의 스키마를 나타내며 컴포넌트의 상태 스키마는 [그림8]과 같이 정의된다.

Component (name, structure, principle_obj, Interface, Internal_obj)	
name	컴포넌트에 대한 유일한 이름
structure	내부 객체들과 그들간의 관계 structure = (Obj, →)
principal_obj	principal_obj ∈ Interface
Interface	Interface ⊆ Obj 인터페이스 객체들의 집합
Internal_obj	Internal_obj = Obj \ Interface

[그림8] 컴포넌트 상태 스키마

Conector Definition Schema
type definition
state schema
operation schema

[그림9] Connector Definition Schema

[그림9]은 커넥터 정의 스키마를 나타내며 커넥터의 상태 스키마는 [그림10]과 같이 정의된다.

Connectors (LinkRules, con_type)	
LinkRules	커넥터와 커넥터, 커넥터와 컴포넌트 사이의 결합 규칙들의 집합
con_type	컴포넌트 통신, 협상을 캡슐화한 추상화 단위

[그림10]커넥터의 상태 스키마

Structure Definition Schema
type definition
operation schema

[그림11] Structure Definition Schema

[그림11]은 구조 정의 스키마를 나타내며 구조 상태 스키마는 [그림12]와 같이 정의된다.

Structure (Inner_com, Inner_con, Link)	
Inner_com	구조내에 포함된 컴포넌트들의 집합
Inner_con	구조내에 포함된 커넥터들의 집합
Link	구조내에 포함된 결합들의 집합

[그림12] 구조 상태 스키마

4. 결론 및 향후 연구방향

소프트웨어 아키텍처는 시스템의 구조적 기술로서 시스템을 구성하는 컴포넌트와 그 컴포넌트들 사이의 상호작용을 기술한 것으로 CBSE에서는 핵심 기술 중 하나로 자리잡고 있다.

본 논문에서는 컴포넌트 명세를 위한 3가지 전략을 제안했다. 첫째, 정형 언어를 사용하여 컴포넌트를 명세한다. 둘째, 커넥터를 기본으로 하는 ADL을 확장하여 컴포넌트 내부의 계산과정을 표현한다. 셋째, 컴포넌트의 종류별로 컴포넌트를 명세한다.

또, 컴포넌트 명세 방법을 제안했다. 첫째 시스템 모델로부터 시스템 타입 명세를 하고, 둘째 시스템 타입 명세로부터 컴포넌트 타입 명세를 하고, 셋째 컴포넌트 타입 명세로부터 객체 타입 명세를 한다.

3가지 컴포넌트 명세 전략과 3단계 명세 방법을 이용하여 컴포넌트를 명세함으로써 ADL과 Formal Methods의 장점을 이용하여 컴포넌트 명세의 정확성을 도모할 수 있다.

그러나, 본 논문에서 제안한 전략 및 방법만으로 완벽한 컴포넌트 명세가 이루어 질 수 없으며, 컴포넌트를 정확하게 표현할 수 있는 컴포넌트 정형 명세 언어에 대한 연구, 시스템 모델을 컴포넌트 명세로 자동화 할 수 있는 도구의 개발, 그리고 컴포넌트 명세의 완벽함을 검증할 수 있는 연구가 필요하다.

참고문헌

[1] O. C. Kwon, S. J. Yoon and G. S. Shin, “CBD Environment: An integrated model of OO techniques and other techniques”, Submitted to the 2nd International Workshop on CBSE held in Conjunction with the ICSE99, Los Angeleles, USA

[2] 김태효, 정인복, 배두환, 차성덕, “소프트웨어 아키텍처의 연구 동향”, 정보과학회 소프트웨어공학회지, 제12권 제3호, 9월 1999년, pp19-30

[3] M. D. Mcilory, "Mass-Produced Software Components", Software Engineering Concepts and Techniques, Van Nostrand Reinhold, 1976

[4] W.F.Ogden, et al, "Special Feature:Component Based Software Using RESOLVE", ACM SIGSOFT, Software Engineering Notes, 19(4), 1994

[5] J.Magee, et al, "Specifying Distributed Software Architectures", Proc. of ESEC95, Sep 1995

[6] M.Shaw, et al, "Abstractions for Software Architecture and Tools to Support Them", IEEE Transactions on Software Engineering, 1995

[7] D.Luckham, et al, "Specification and analysis of system architecture using Rapide", IEEE Transactions on Software Engineering, 21(4), Apr 1995

[8] D.Garlan, et al, "Exploiting Style in Architectural Design Environments", Proc. of SIGSOFT '94 Symposium on the Foundations of Software Engineering, Dec 1994

[9] D.Garlan, R.T.Monroe, and D.Wile, "Acme: An Architecture Description Interchange Language", Procs. of CASCON '97, Nov 1997

[10] M. Moriconi, X. Qian, and R.A.Riemenschneider, "Correct Architecture Refinement", IEEE Transactions on Software Engineering, 21(4), 1995

[11] R.J. Allen, "A Formal Approach to Software Architecture", Carnegie Mellon University, CMU-CS-97-144, 1997

[12] 우치수, 김갑수, 이명재 공역, “소프트웨어공학”, 홍릉과학출판사, 1998, pp163-216

[13] A. Harry, "Formal Methods Fact File:VDM and Z", John Willey & Sons, 1996, pp173-258

[14] S. Stepney, R. Barden, and D. Cooper, "Object Orientation in Z", British Computer Society, 1992, pp59-77